Prophet: Proactive Candidate-Selection for Federated Learning by Predicting the Qualities of Training and Reporting Phases

5 authors, including:

Huawei Huang
Sun Yat-Sen University
61 PUBLICATIONS   **574** CITATIONS

SEE PROFILE

Kangying Lin
Sun Yat-Sen University
2 PUBLICATIONS   **0** CITATIONS

SEE PROFILE

Song Guo
The Hong Kong Polytechnic University
497 PUBLICATIONS   **7,070** CITATIONS

SEE PROFILE

Zibin Zheng
Sun Yat-Sen University
246 PUBLICATIONS   **6,641** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Space-Terrestrial Integrated Networking View project

Scalability of Blockchain View project

Prophet: Proactive Candidate-Selection for Federated Learning by Predicting the Qualities of Training and Reporting Phases

Huawei Huang[*], Kangying Lin[*], Song Guo[‡], Pan Zhou[†], Zibin Zheng[*]

[*]School of Data and Computer Science, Sun Yat-Sen University, Guangzhou, China
[†]School of Cyber Science and Engineering, Huazhong University of Science & Technology, Wuhan, China
[‡](Corresponding) Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China
huanghw28@mail.sysu.edu.cn; linky23@mail2.sysu.edu.cn; song.guo@polyu.edu.hk; panzhou@hust.edu.cn; zhzibin@mail.sysu.edu.cn

Abstract—Federated Learning (FL) is viewed as a promising technique for future distributed machine learning. It permits a large number of mobile devices participating in the training of a global model collaboratively without having to expose their local private data. Although the challenge of the network connection will be much relieved in 5G/B5G era, the training latency is still an obstacle preventing FL from being largely adopted. One of the most fundamental problems that leads to large training latency is the bad *candidate-selection* of FL participants. To the best of our knowledge, the existing candidate-selection algorithms belong to the *reactive* manner. Under such *reactive* selection, the FL parameter server only knows the currently-observed resources of all candidates. In the dynamic FL environment, the mobile devices selected by the *reactive* candidate-selection algorithms very possibly fail to complete the *training* and *reporting* phases of FL. To this end, we study the *proactive* candidate-selection for FL in this paper. We first let each candidate device locally predict the qualities of both its *training* and *reporting* phases using the LSTM network. Then, the proposed *candidate-selection* algorithm is implemented by the Deep Reinforcement Learning (DRL) framework, which can adapt to the dynamically varying factors in the metropolitan edge computing environment. Finally, the real-world trace-driven experiments prove that the proposed *proactive* approach outperforms the existing *reactive* algorithms with respect to the ratio of valid participants and the test accuracy of the aggregated global FL model.

Index Terms—Federated Learning, Proactive Candidate-Selection, Deep Reinforcement Learning

I. INTRODUCTION

Federated Learning (FL) [1], [2] is a branch of distributed machine learning that enables a group of distributed devices to train their individual local models using the local dataset. Thus, FL is a promising computing paradigm in our future intelligent life, especially under the fifth generation (5G) and the beyond (B5G) communications networks. For example, the *FederatedAveraging* (FedAvg) algorithm [1] can help mobile users predict the next-words when users are using the Google's GBoard [3] in their smartphones.

To realize a large-scale federated learning framework, a number of challenges must be addressed. For example, to attract a large number of candidate devices participating into the federated learning, the communication latency and network bandwidth between each distributed device and the FL parameter server are viewed as two typical bottleneck resources.
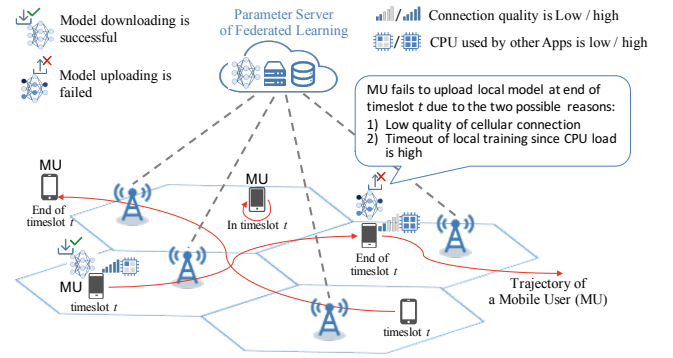

Fig. 1. Candidate selection for Federated Learning based on the predicted qualities of both network connections and computing capabilities within a training round (measured in timeslot).

Although those two bottleneck resources will be much relieved in 5G/B5G era, another particular challenge is to handle the highly dynamic mobility of mobile users. The uncertainties include the dynamic trajectories of mobile users, the individual daily usage habits of their smartphones, unexpected cellular network fluctuations, etc. The mobility of mobile users in a metropolitan area and the user's usage-habit of their smartphones are difficult to predict accurately. Even though a group of candidate devices with good communication and computing resources are selected to participate in a federated learning at a timeslot, no one can ensure they can complete the local training and upload the updated model timely to the FL parameter server in the near future.

Recently, Google's team [2] has implemented a scalable federated-learning production system by exploiting mobile devices. According to their technique report [2], each round of synchronous-fashioned federated learning include 3 major phases: (1) the selection of candidate devices; (2) the configuration on mobile devices and local training; and (3) the reporting of local training results.

Candidate-selection as the first essential phase, determines what group of candidate devices can be chosen to participate in an FL training task. To select the most appropriate group of mobile devices for an FL task is still an open issue, especially

when the number of devices reaches billions in the near future.

To the best of our knowledge, the existing candidate-selection algorithms [1], [2], [4] for distributed federated learning can be classified into the *reactive* manner. That is, candidate devices are selected only by their currently-observed resources, such as the network-connection quality and the remaining computing capability. However, we argue that the candidate-selection of FL only depending on the currently-observed resources is short-sighted. For example, if a device with good resources of both connection and computing was selected at the *selection* phase, but it may turn to have a low computing capability during the *training* phase, or have a low quality of network connection during the *reporting* phase. Those two situations can bring failures to a round of FL training. As a result, all the previous configurations that have been devoted to this mobile device are in vain.

Therefore, in this paper, we aim to propose an efficient proactive algorithm that can perform a *prophetic* selection for federated learning by predicting mobile users' future qualities of both training and reporting phases. Our proposed approach mainly includes two parts. The first part is the prediction of users' patterns with respect to the mobility trajectory and the App-usage habits in their smartphones. The second part is a decision-making scheduling algorithm that is based on deep reinforcement learning (DRL) framework.

The contributions of this paper are summarized as follows.

- Firstly, different from the existing *reactive* approaches, we study the *proactive* candidate-selection for Federated Learning by predicting the qualities of both the *training* and *reporting* phases.
- Secondly, to solve the proposed profit-maximization problem, we have designed a DRL-based algorithm, which is able to endure the highly uncertain events in the dynamic FL environment.
- Finally, we developed a smartphone App to collect trace data for our experiments. The real-world trace dataset was then collected from our campus for 6 weeks. The trace-driven experimental results show that the proposed *proactive* candidate-selection algorithm outperforms the state-of-the-art baselines.

The remaining of this paper is organized as follows. Section II reviews the state-of-the-art studies. Section III describes the system model and problem formulation. Section IV elaborates the prediction scheme of the proposed approach. Then, Section V discusses the DRL-based candidate-selection algorithm. Section VI shows the real-world trace-driven experiments. Finally, Section VII concludes this paper.

II. RELATED WORK

A. Proactive Provisioning based on Sequence Prediction

Understanding human mobility [5] is a classic scientific problem. A lot of proactive service provisioning can be conducted by predicting the human's mobility. The most popular approach used to predict mobility is based on Recurrent Neural Networks (RNN), such as Long Short Term Memory (LSTM)

[6]–[8] and Gated Recurrent Unit (GRU) [6], [9]. Several representative studies are reviewed as follows. Feng *et al.* [6] proposed DeepMove, in which authors aim to capture the complicated sequential transitions during human mobility based on a multi-modal embedding RNN. Zhou *et al.* [7] studied the cost-efficient server provisioning problem, which aims to proactively schedule the activating/deactivating of servers in edge computing environments. The proactive schedule is achieved by predicting the arriving data streams using an online learning algorithm. Next, aiming to predict the activity and the location of mobile users, Liao *et al.* [9] proposed a particular spatial activity topic modeling, by adopting the multi-task context into their RNN based prediction approach. To forecast the citywide crowd events, Jiang *et al.* [8] proposed a LSTM-based multitask encoder-decoder framework, in which a data-driven dynamics prediction intelligent system has been built.

In addition, Hidden Markov Model (HMM) is another popular method to modeling personalized sequences such as users' mobility and living habits. For example, based on a large-scale mobility dataset collected from LTE networks in a Southern Chinese city, Lv *et al.* [10] investigated the spatiotemporal prediction and users' next destination by employing the HMM based predictor. Shi *et al.* [11] studied the personalized sequential patterns based on a state-sharing sparse HMM model under the situation of data scarcity.

Taking the advances of LSTM when predicting spatiotemporal sequences into account, we also exploit the LSTM-based prediction approach in the proposed proactive candidate-selection algorithm.

B. Candidate Selection for Federated Learning

As the pioneer studies of federated learning, McMahan *et al.* [1] and Bonawitz *et al.* [12] introduced the *FedAvg* algorithm. For example in [1], authors proposed a communication-efficient learning framework that selects the participating devices to train a global model using the decentralized rich local dataset. The FL server then conducts the model averaging with the local stochastic gradient descent (SGD) reported by each mobile device.

Based on the original FL protocol *FedAvg* described in [1], Nishio *et al.* [13] proposed a new FL protocol named *FedCS*, which towards the efficient FL by employing heterogeneous mobile users in a mobile edge computing environment. Particularly, the candidate selection in such *FedCS* is performed by requesting the resource information of mobile users at real time. Thus, *FedCS* protocol is essentially the *reactive* manner.

Later, Chen *et al.* [4] studied the implementation of FL framework for wireless networks. To jointly consider the candidate selection and resource allocation while minimizing the FL loss function, authors first formulated an optimization problem. Then, they derived a closed-form expression of the anticipated convergence rate of the FL algorithm, aiming to quantify the effect of wireless factors on FL training.

Google's team [2] has recently proposed a technique report describing their deployment experience of large-scale

FL architecture. However, in their candidate-selection phase, the participants are chosen by a simple *reservoir-sampling* approach. Therefore, the new FL protocol proposed by [2] has a large freedom to design a more sophisticated selection scheme with respect to a certain bias.

Through the review of the state-of-the-art selection approaches, we find out the two following facts.

- To the best of our knowledge, the existing candidate-selection algorithms for FL belong to the *reactive* manner. That is, the candidate-selection over all mobile participants is performed only when receiving their current device information. Thus, the *reactive* manner incurs the inevitable resource-requesting latency. Moreover, the *reactive* manner can only see the computing environment of the current timeslot. That is why we say this manner is *short-sighted*.

- The existing studies usually consider that all mobile users have the same usage-behavior pattern, thus they simulate the state of CPU and network connection of mobile devices by exploiting statistical distribution models such as *Poisson* and *Uniform* distributions. However, this simple assumption could be unrealistic for capturing the real-world users' behaviors in a highly uncertain computing environment.

In contrast, our proposed approach is a *proactive* manner, under which the candidates are selected by the predicted qualities of both the training and reporting phases. In addition, the traces of mobile users used in our experiments were collected from the real volunteers for 6 weeks. The real trace of mobile users provides us a lot of confidence when designing our *proactive* candidate-selection algorithm.

III. System Model and Problem Formulation

A. System Model

As illustrated in Fig. 1, we consider a mobile edge computing (MEC) environment, which is built on metropolitan cellular networks. The synchronous-fashioned computing architecture consists of a central FL parameter server, a group of base stations, and a large number of mobile devices with heterogeneous resources. The central FL parameter server initializes a global model for a specific computing task such as image classification. For each training task, the FL server recruits a group of mobile users (MUs) to join in a training task. The selected MUs are viewed as FL *participants*, who strive for training a global model collaboratively. All MUs willing to join a specific FL task are denoted by the set $\mathcal{N} = \{1, 2, 3, ..., N\}$.

Generally, each round of the synchronous FL training [2] includes three major phases: 1) *selection*, 2) *training*, and 3) *aggregation*, i.e., model *reporting*. At the beginning of each selection phase, each MU sends a participating-request message to the FL server. This request message consists of their current resource information on their device (e.g., CPU utilization, state of wireless connection). To conveniently describe the execution of FL system, we also call each round

TABLE I
Symbols and Notations

$\mathcal{N}$	the set of all candidate devices for FL
K	the maximum number of selected participants for each FL round
$\mathcal{T}$	the set of FL training rounds (i.e., all timeslots)
$q_i(t)$	currently-observed connection quality of MU i at timeslot t
$q_i^*(t)$	predicted connection quality of MU i within round t
$c_i(t)$	currently-observed available CPU capacity of MU i at timeslot t
$c_i^*(t)$	predicted available CPU capacity of MU i within round t
$e_i(t)$	a binary indicator that indicates whether device i uploads its updated model parameters successfully to FL server at the end of reporting phase
$X_i(t)$	a binary variable that indicates whether candidate device i is selected by the FL server at round t

of FL training a *timeslot*, which is illustrated in Fig. 2. All timeslots are recorded in set $\mathcal{T} = \{1, 2, 3, \cdots, T\}$. Thus, the deadline for each round of FL training is actually the length of a timeslot. In a real experimental setting, the length of a timeslot could be measured in milliseconds.

Assuming that MU i has a number of CPU capacity, which represents its currently available computing resource that can be applied to FL training. The network connection quality of each MU is measured by the cellular parameters which can be perceived by each smartphone.

When the number of the above-mentioned candidate MUs reaches a certain amount, the FL server begins to select participants, and sends the global model to these selected participants. A total deadline T will be set for all training rounds. Due to the limitation of the wireless bandwidth and the deadline for each round, it is not a good idea to select as many participants as possible. Therefore, the FL server must select an appropriate subset of all candidate MUs at the very beginning of an FL training task. A good candidate selection not only can maximize the training efficiency, but also can avoid to wait too long for the slow-training participants, which fail to complete their local training and/or upload the local models before the deadline of a round.

Particularly, due to the mobility of these MUs, they may traverse different locations at different timeslots such that the quality fluctuations of cellular connection between MUs and the FL server are inevitable. Meanwhile, each of these MUs may use different applications (APPs) at different timeslots, such as playing high-computation games or watching high-resolution online videos, resulting in the dynamic changes of the remaining computing resources (i.e., the available CPU capacity) for the FL tasks.

As we can see from Fig. 1, not all participants can complete their local training when a candidate is selected. This is because only the current training resources can be observed at the beginning of each round (Fig. 2). Due to the highly-dynamic mobility of MUs in the MEC environment, the qualities of both network connections and available computing capabilities could be very different at the end of each round
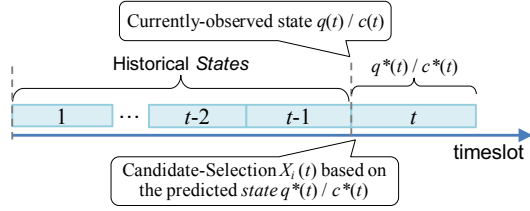
Fig. 2. The system model when deciding to select the candidate $i \in \mathcal{N}$.

of FL training. Therefore, if there exists an algorithm that can predict the qualities of those two factors, a good group of candidates can be selected in a *proactive* manner. Accordingly, we use $q_i^*(t)$ and $c_i^*(t)$ to denote the predicted qualities of network connection and CPU capacity of MU i within timeslot t, respectively.

The other important symbols and notations are explained in Table I.

B. Problem Formulation

As shown in Fig. 2, to denote whether a candidate device $i \in \mathcal{N}$ is selected or not by the FL server at the beginning of each timeslot, we define the following binary variable

$$X_i(t) = \begin{cases} 1, & \text{if FL server selects device } i \text{ at timeslot } t, \\ 0, & \text{otherwise.} \end{cases}$$

To represent whether a chosen candidate, i.e., a *participant*, successfully uploads its updated model to the FL parameter server or not, we then define a binary indicator $e_i(t)$, the definition of which is shown as follows.

$$e_i(t) = \begin{cases} 1, & \text{if participant } i \text{ successfully uploads updated} \\ & \text{model to the FL server within timeslot } t, \\ 0, & \text{otherwise.} \end{cases}$$

We then have the following formulation of the problem **Prophet**: proactive candidate-selection by predicting the qualities of training and reporting phases.

$$\textbf{Prophet} : \max \sum_{i \in \mathcal{N}} X_i(t)e_i(t) \qquad (1)$$

$$\text{s.t.} \sum_{i \in \mathcal{N}} X_i(t) \leq K, \quad \forall t \in \mathcal{T} \qquad (2)$$

Variables: $X_i(t) \in \{0, 1\}, \forall i \in \mathcal{N}, \forall t \in \mathcal{T}.$

We call a participant a *valid* one if it can successfully complete the training and reporting its updated local model to the FL server within a timeslot. Thus, the objective function (1) claims that the total number of *valid* participants should be maximized for each FL round $t \in \mathcal{T}$. Constraint (2) defines the maximum number, i.e., K, of all participants that are selected by FL server for each round of training. Note that, to solve this problem in a proactive manner, $e_i(t)$ must be predicted using a certain prediction approach.

IV. PREDICTING THE NEXT-STATE OF CANDIDATES

This section presents the prediction towards the *next-state*, i.e., the qualities of network connections and computing capabilities within a timeslot, of mobile devices based on the Long Short-Term Memory (LSTM) network. The predicted *next-state* is the foundation of the proposed *proactive* candidate-selection algorithm.

There are three important factors that are believed as the bottleneck to a participant when training a task in an FL round, i.e., the size of heterogeneous local data, computing capabilities and network conditions [13]. Here, we assume that the data size of each mobile device is relatively stable during the training of an FL task. Consequently, the major prediction objectives are the other two factors, i.e., the computing capabilities and network connections in the *reporting* phase, as the next-state of mobile devices.

A. Local Prediction for the Next-State of Each MU

When deciding on what component to perform the *next-state* prediction, in fact we have two choices: 1) let all mobile devices report their local trace data to a central server, the server then predicts the next-state of all MUs; or 2) let the mobile devices predict their individual next-state using their local dataset. Taking the users' privacy into account, it is unsuitable, sometimes illegal, to aggregate such sensitive trajectory data in a central server. Furthermore, when the number of MUs reaches a large scale, the central server will not be capable to predict the next-state for each MU, if adopting the first choice. Thus, we adopt the second choice, which is shown in Fig. 8. Each mobile device executes the LSTM-based local prediction algorithm to predict its own *next-state*, which is then fed back to the candidate-selection algorithm.

B. Trace Collection using a Self-Developed App

To collect the trace dataset of mobile users, we first developed an Android App named *Tracer*. This App together with the technique report of the dataset collection are open to download from the homepage of our lab http://www.xintelligence. pro/datasets. We then recruited about 100 students as volunteers who helped us collect their daily routines and App-usage habits on smartphones. All trace datasets were collected in the campus of Sun Yat-Sen University (Guangzhou) for over 6 weeks. Each volunteer's smartphone reported the collected data to the central server located at our lab everyday. In detail, the dataset mainly includes three categories:

- The GPS data, i.e., the latitude and longitude, of each MU, which is used to calculate the Place of Interests (POIs) [10] of candidates.
- The cellular parameters of channel signals, i.e., the *RSSI, RSRP, RSRQ, SINR* perceived by each smartphone. These signal parameters are used to quantify the network connection quality of each mobile device.
- The CPU workload consumed by the top 10 Apps in each mobile device. This type of data is used to infer the available computing capability that can be spent on the local Federated Learning task on each candidate device.

```
Latitude,Longitude,Time,RSSI,RSRP,RSRQ,SINR,CPU-Consumed(%)
23.058937,113.392689,2019-10-15 10:03:07,-69,-79,-7,23.8,83.0
23.058958,113.392704,2019-10-15 10:04:07,-69,-79,-7,23.8,76.0
23.058958,113.392706,2019-10-15 10:05:07,-69,-79,-7,23.8,71.0
23.058941,113.392693,2019-10-15 10:06:07,-69,-79,-7,23.8,76.3
23.058969,113.392694,2019-10-15 10:07:07,-69,-79,-7,23.8,6.6
23.058963,113.392692,2019-10-15 10:08:07,-75,-81,-6,22.8,7.3
23.058971,113.392692,2019-10-15 10:09:07,-75,-81,-6,22.8,14.6
```

Fig. 3. Raw trace collected from our campus by one of the 100 volunteers. The raw trace includes *latitude*, *longitude*, *time*, *cellular parameters*, and *CPU workload* consumed by the top 10 Apps in a smartphone.
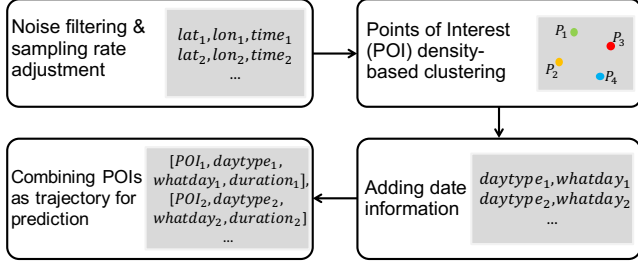

Fig. 4. Pre-processing the GPS traces into the trajectories of mobile users.

(*RSSI, RSRP, RSRQ, SINR*): a tuple of connection-quality parameters of cellular networks
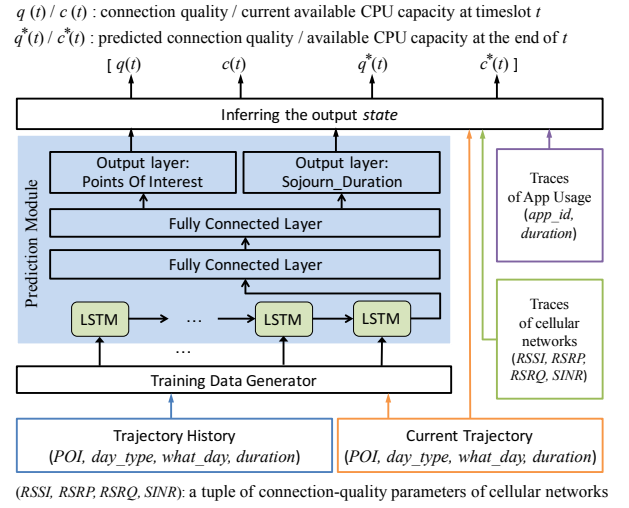
Fig. 5. The framework of LSTM-based local prediction towards the *next-state* of each mobile device.

Fig. 6. POIs of all candidate devices located in our campus.

We only show a part of raw trace data in Fig. 3 due to space limitation.

C. Pre-processing of Trajectory Traces

The raw datasets must be pre-processed, then can it be exploited to predict user's *next state* in a timeslot t.

First, each mobile user collects its raw GPS data using the smartphone App *Tracer*. Every raw GPS trace is represented by (lat_i, lon_i, t_i), which record the user's position of latitude lat_i and longitude lon_i at time t_i. However, the raw trace data contains some noises because of the inherent signal deviation of GPS. To retrieve the trajectories of MUs that can be directly provided to the prediction algorithm, we perform the pre-processing according to Fig. 4. The 4 main steps are as follow.

1) Filtering the GPS noises and adjusting the sampling rate over all trace data items.
2) Using the GPS data after filtering in the previous step, step 2 is to cluster the POIs of mobile users based on the density of their daily routines. Fig. 6 shows the results of all POIs generated by this step based on the datasets collected from all volunteers in our campus.
3) Adding additional date information such as the day type and what day during a week.
4) The final step is to combine POIs with the additional date information achieved in step 3. After this combining, the trajectory of each mobile user is obtained.

Please refer to our technique report [14] for more details of data pre-processing.

D. Local Next-State Prediction

Fig. 5 shows the framework of *state* prediction. Here the state includes the qualities of network connections and the available CPU capacity of each candidate device. When the trajectory history is generated according to the previous pre-processing, a LSTM-based prediction model is used for the prediction module. In this module, we use a many-to-one architecture to learn each candidate's mobility pattern from their trajectory histories.

To satisfy the format requirement of LSTM network, all trajectory data should be converted by the *training data generator* module. In this data generator, the *duration* of a mobile user staying in a location during his/her trajectory is changed from the original data format (year-month-day hour:minute:second) into the unix timestamp format. We then use the min-max scaling approach to standardize the *duration* attribute. After standardization, each trajectory item is converted to [*POI*, *day_type*, *what_day*, s_t, e_t], in which *day_type* represents that whether a day is a working day, weekend, or holiday; *what_day* stands for the week-day; while s_t and e_t denote the starting and ending time when a candidate device arrives at and leaves from a POI, respectively.

With the data processed by the *training data generator* module, we then use two *fully connected layers* to predict the mobile users' location-based information in a timeslot t. The output layers of the prediction module will yield the two
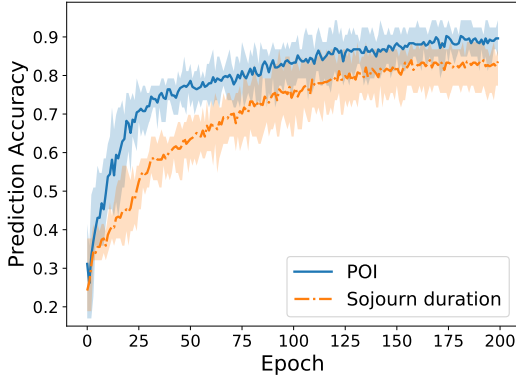
Fig. 7. Accuracy of prediction results using LSTM w.r.t *POI* and *Sojourn duration* of candidate devices.

prediction results, i.e., the *POI* and the *Sojourn duration* of each mobile user.

The *POIs* and *Sojourn duration* of mobile users, combining with the *Traces of App Usage* and the *Traces of cellular networks*, are used to infer the the output *state* of candidates, i.e., the tuples of $[q(t), c(t), q^*(t), c^*(t)]$. Due to space limitation, we elaborate how to infer such the *state* of candidates in our technique report [14].

Finally, Fig. 7 illustrates the prediction accuracy of the LSTM-based model with respect to (w.r.t) *POI* and *Sojourn duration*. Within 200 epochs, we see that the accuracies of *POI* and *Sojourn duration* reach 90% and 82% on average, respectively. Although the accuracies are not so high, the proposed proactive candidate-selection algorithm can still have the highest FL training results compared with other baseline algorithms. The performance comparison is presented in Section VI.

V. DRL-BASED CANDIDATE-SELECTION ALGORITHM

Although the network state in the near future can be predicted with a high accuracy, some significant uncertainties can still happen quite often in a dynamic federated learning environment. For example, the failures of cellular networks may occur during disasters, and the mobile users may suddenly become offline. Thus, to handle the unexpected dynamic events that can not be accurately predicted, we propose to design a Deep Reinforcement Learning (DRL)-based algorithm. Generally, the reinforcement learning [15] approach is a trial-and-error search, in which the agent not only prefers to exploit the good actions that have already been tried in the past, but also to explore new actions that may indicate higher rewards in the future. Building the policy network using the deep neural network (DNN), the reinforcement learning turns to be DRL. In particular, our proposed DRL-based algorithm is implemented by the Double DQN (DDQN) framework [16]. In the following, we describe how to construct such the proposed DDQN-based algorithm.
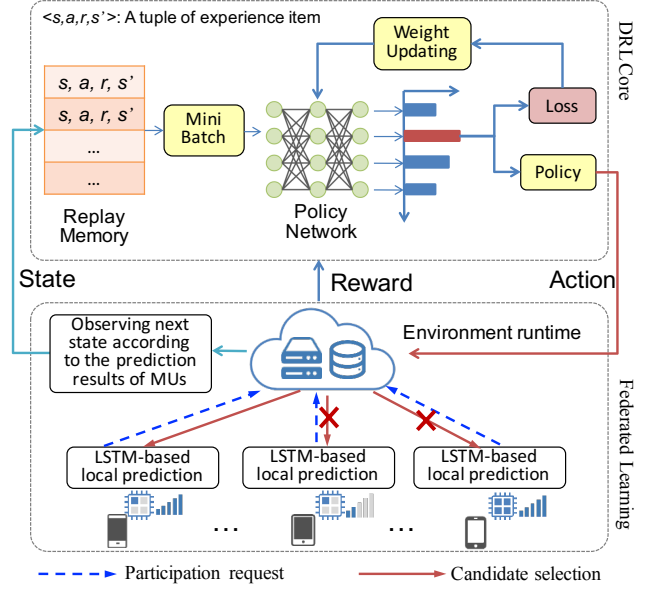
A. Overview of DRL-based Framework

Fig. 8. Interactions between *DRL Core* and *Federated Learning* modules.

As Fig. 8 shows, the proposed DRL-based Framework mainly includes two modules: 1) the *DRL Core*, and 2) the *Federated Learning*. In DRL Core, the DDQN-based algorithm contains an offline constructing phase of deep neural network that can approximate the action-value function with corresponding states and actions, an online dynamic deep Q-learning phase for action filtering, and the dynamic network updating. For each round of FL, the role of DRL Core is to conduct the candidate-selection for Federated Learning module in a proactive manner.

In Federated Learning module, each mobile device provides its *next-state* to the FL server. All the predicted *state* results can be observed by DRL Core as *experience* items. Such the experience items are aggregated in the Replay Memory of DDQN algorithm. Via the training of DRL framework, a good policy will be yielded as the *Action*, which is then treated as a solution to the candidate-selection in the FL server. In detail, the training algorithm of the Q-network is presented in Algorithm 1.

B. Problem Reformulation based on DRL

In DRL Core, the DDQN-based algorithm is essentially solving a stochastic optimization problem. This problem can be represented by $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}(t) \rangle$, where $\mathcal{S}$, $\mathcal{A}$ and $\mathcal{R}(t)$ ($\forall t \in \mathcal{T}$) are the *state space*, *action space*, and *reward function* of the DDQN-based algorithm, respectively. We then define them associated with our problem as follows.

State Space: Combining all the states of N candidates collected by the FL server at the beginning of round t, the state space is denoted by $\mathcal{S}(t) = \bigcup_{i \in \mathcal{N}} S_i(t)$, where $S_i(t)$ is the state-tuple of each candidate $i \in \mathcal{N}$. Recall that each candidate's state is the output of the local LSTM-based

---

Algorithm 1: DDQN-based Proactive Candidate-Selection

---

1 **Initialize** evaluation and target Q-network parameters with $\boldsymbol{\theta}$, $\boldsymbol{\theta}^-$;

2 **Initialize** replay memory with the size of X.

3 **for** *episode* $e = 1$ *to* E **do**

4   **for** *round* $t = 1$ *to* T **do**

5   N candidates send participant requests to FL server;

6       $S_i(t) \leftarrow [q_i(t), c_i(t), q_i^*(t), c_i^*(t)], \forall i \in \mathcal{N}$; // FL server observes states

7   Select action a according to the ϵ-greedy policy;

8   Execute action a: choose K candidates to distribute FL model;

9   Observe reward r and the state s' of the next timeslot $t+1$;

10   Update Reply Memory: $\mathcal{M} \leftarrow \mathcal{M} \cup \{\langle s, a, r, s' \rangle\}$;

11   Sample random mini-batch of B experience items from $\mathcal{M}$;

12   **for** $j = 1$ *to* B **do**

13        $a^{\max} \leftarrow \arg\max_{a' \in \mathcal{A}} Q(s_j', a'; \boldsymbol{\theta})$;

14        $y_j \leftarrow r_j + \gamma Q\left(s_j', a^{\max}; \boldsymbol{\theta}^-\right)$;

15      $L(\boldsymbol{\theta}) \triangleq \frac{1}{N} \sum_{i=1}^N (y_i - Q(s_i, a_i; \boldsymbol{\theta}))^2$;

16   Perform a gradient descent step on $L(\boldsymbol{\theta})$ to update $\boldsymbol{\theta}$;

17   Reset: $\boldsymbol{\theta}^- \leftarrow \boldsymbol{\theta}$ in every C steps

---

prediction model shown in Fig. 5. Thus, the state of candidate $i \in \mathcal{N}$ observed by the FL server is denoted by:

$$S_i(t) = [(\mathbf{q}_i(t), \mathbf{c}_i(t), \mathbf{q}_i^*(t), \mathbf{c}_i^*(t))], i \in \mathcal{N}, \forall t \in \mathcal{T}.$$

Therefore, we have the *state* space $\mathcal{S} = \bigcup_{t \in \mathcal{T}} \mathcal{S}(t)$. The s and s' appeared in Fig. 8 and Alg. 1 are short for $\mathcal{S}(t)$, $\mathcal{S}(t+1)$, respectively.

Action Space: When observing the state set $\mathcal{S}(t)$, DRL Core will make the candidate-selection decisions for FL server. The FL server then distributes the global model to each participant for training using the local datasets. The action space of the candidate-selection in timeslot t is defined as:

$$\mathcal{A}(t) = \{\mathbf{X}_i(t), \ i \in \mathcal{N}\}, \forall t \in \mathcal{T},$$

where $\mathbf{X}_i(t) = 1$ or 0 means that the candidate i is selected to participate in FL round t or not. Note that, the constraint (2), i.e., $\sum_{i \in \mathcal{N}} \mathbf{X}_i(t) \leq K$, must be satisfied. Similarly, we have the *action* space $\mathcal{A} = \bigcup_{t \in \mathcal{T}} \mathcal{A}(t)$.

Reward Function: Based on the original objective function (1), we design a reward function for the *Prophet* problem. The reward function is defined as follows:

$$\mathcal{R}(t) = \sum_{i \in \mathcal{K}} \alpha_i \mathbf{X}_i(t) e_i(t) - \beta \sum_{i \in \mathcal{K}} \mathbf{X}_i(t) \overline{e}_i(t), \forall t \in \mathcal{T}, \quad (3)$$

where $\mathcal{K}$ ($|\mathcal{K}| = K$) is the set of the candidates randomly chosen from $\mathcal{N}$. Particularly, $\overline{e}_i(t)$ represents the opposite situations of $e_i(t)$, i.e., when $e_i(t) = 1$, $\overline{e}_i(t) = 0$, and vice

versa. We see that the reward contains two terms. The first term $\sum_{i \in \mathcal{K}} \alpha_i \mathbf{X}_i(t) e_i(t)$ is a positive reward, which is essentially the objective (1). Then, the second term $\beta \sum_{i \in \mathcal{K}} \mathbf{X}_i(t) \overline{e}_i(t)$ is a *timeout* penalty, which is used to quantify the impact brought by the participants who failed to complete the FL training task within timeslot t. As mentioned, the reasons behind the failure include two situations, i.e., a participant failed to upload the updated model to FL server due to low-quality cellular connections, or a participant took too much training time because of its low computing capability. Thus, the parameter α_i is a scale factor that decreases as the model-update time of the valid participant i increases, while β is the negative scale factor of the *timeout* penalty.

VI. PERFORMANCE EVALUATION

A. Methodologies and Settings

Basic Settings: In experiment settings, we set totally $N=100$ mobile devices as the candidates for federated learning. At most $K=10$ of them will be selected to participate in each round of federated learning. At the beginning of each round, all candidates report their device information, which includes their individual network connection quality and current computing capability, to the FL parameter sever. Next, the parameter server selects the best group of devices as the *participants* for training an FL task, according to different candidate-selection algorithms. Under the proposed DDQN-based algorithm, after a number of E episodes of training with other parameters configured following Table II, the *DRL Core* will learn a good policy for candidate selection. This good policy is then provided to the FL server to guide the distribution of global FL model.

Settings for Policy Training: The trace dataset of each of the K candidates is divided into two parts. The first part is for training the DDQN-based model, while the other part is to provide the trajectory for a mobile device during the trace-driven experiments of online FL training. During the training of DDQN model towards the best candidate-selection policy, the K candidates chosen by the FL server will not be altered until a good policy is retrieved within a certain number of episodes.

Settings for Global FL Training: To train the global Federated Learning model, we choose two well-known image datasets MNIST [17] and CIFAR-10 [18]. According to [1], the distribution of data samples in a dataset substantially affects the training quality of distributed learning. Considering that the image dataset in our smartphone is in fact following the non independently identically distribution (non-i.i.d.). Therefore, in our experiments, we evaluate the test accuracy of algorithms using such the non-i.i.d. datasets.

B. Baseline Algorithms

Totally we compare the proposed DDQN-based algorithm with 4 baselines. The basic idea of each is explained as follows.

FedCS [13]: The FL server chooses participants only by the currently-observed CPU capability and channel conditions
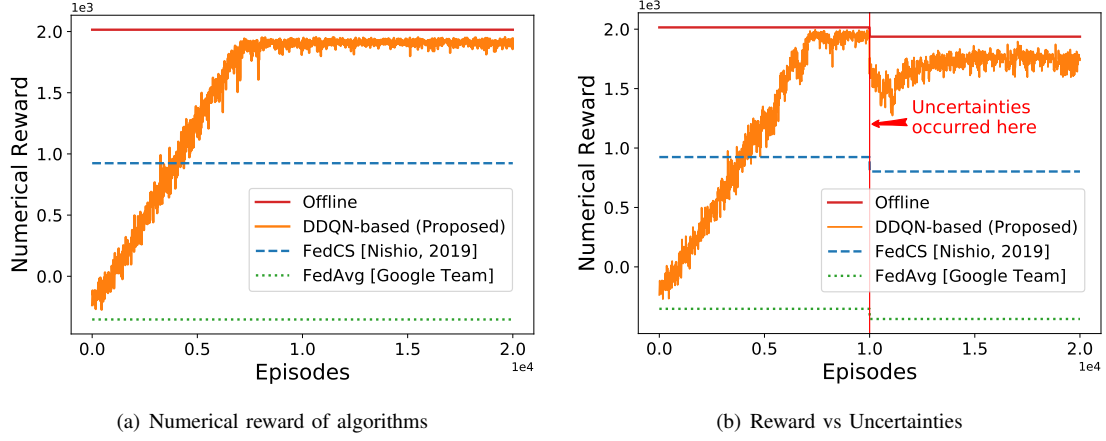
(a) Numerical reward of algorithms (b) Reward vs Uncertainties

Fig. 9. Numerical reward while striving for candidate-selection policy.

Parameters	Value	Parameters	Value
X	8×10^4	B	32
E	2×10^4	γ	0.9
T	200	NN size	$32 \times 32 \times 32$
N	100	Optimizer	Adam
K	10	learning rate	0.001
C	300	ϵ-greedy	0-0.9

of candidates. The better computing capability and network connections observed at current timeslot, the higher chance to be selected by the FL server.

FedAvg [1]: This probably is the most original FL aggregation strategy. Under this algorithm, the FL server chooses K random mobile devices as participants to train each round of FL tasks.

Offline: We also implement an *offline* algorithm, in which the participants are selected in each FL round depending on their trace-driven states throughout all timeslots $t \in \mathcal{T}$. Thus, this *offline* scheme yields the *optimal* solutions.

C. Metrics of Evaluation

Numerical Reward: The numerical reward, which is calculated by equation (3), closely relates to the objective function (1).

Ratio of Valid Participants: Another factor that highly reflects the objective function (1) is the ratio of valid participants selected by algorithms. A higher ratio of the valid participants selected by an algorithm indicates that this algorithm can choose a better group of candidates.

Test Accuracy of Aggregated FL Model: We also evaluate the test accuracy of the aggregated global FL model, which is trained by all valid participants collaboratively.

D. Numerical Reward

Fig. 9(a) shows the numerical rewards of all algorithms. First, we see that *FedAvg* shows the lowest reward, because

all the K=10 candidates were selected by this algorithm to participate into each round of the FL training. However, a large portion of those participants cannot complete their FL training tasks every round due to the unknown dynamics of computing environment. The participants who failed in some rounds brought the negative reward. That is why FedAvg has the lowest numerical reward.

Then, *FedCS* algorithm shows a numerical reward around 950, which is much higher than the reward of FedAvg. This is because FedCS suggests the FL server selecting a number of the best participants during the K=10 candidates, only depending on the currently-observed devices' state $[q(t), c(t)]$. However, this *reactive* manner of selection only sees the state of the current timeslot. The future state of candidates cannot be known currently. Thus, some participants selected probably fail to train the FL task in the subsequent timeslots. The failed participants incur the negative rewards under FedCS.

We now analyze the performance of the proposed DDQN-based algorithm. This algorithm starts to train the candidate-selection model with the states of both the currently-observed and the predicted within each timeslot, i.e., $[q(t), c(t), q^*(t), c^*(t)]$. At the beginning of selection-model training, DDQN randomly selects a random number, which is not greater than K, of candidates to participate into the FL training. Thus, the reward is low at the beginning. Through many times of explorations, DDQN keeps learning from the experience items. The candidate-selection policy also keeps improving. We observe that the reward of DDQN 1) grows higher than that of FedCS at the 3000th episode, and 2) converges to a very high numerical reward around 1900 after the 7500th episode. The final converged reward is very close to that of the *offline* solution. This shows that the DDQN has already learned a near-optimal candidate-selection policy. This good reward proves that the prediction of state $[q^*(t), c^*(t)]$ in the near future within a timeslot t enables FL server to select the group of candidates who have bad observed states currently but will have better states in the subsequent timeslots. With such the proactive prediction, the proposed proactive scheme
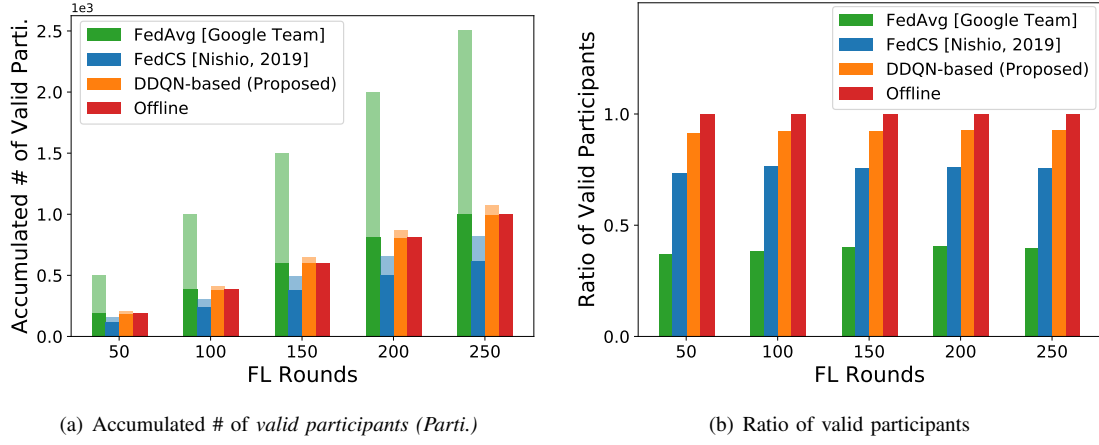
(a) Accumulated # of *valid participants (Parti.)*

(b) Ratio of valid participants

Fig. 10. Performance of *valid participants* during FL training.

can be benefited by the potential group of candidates which bring positive rewards.

E. What Will DDQN Perform under Uncertain Environments?

You are probably wondering what will the algorithms perform under the highly uncertain environments. We show them using Fig. 9(b). The left-hand half of Fig. 9(b) shows that the DDQN has learned a high-reward policy around the 7000th episode. At the 10000th episode, we drastically change the cellular connection parameters manually on purpose to simulate the unexpected uncertainties, e.g., the crash of cellular base stations. Although this change doesn't impact the prediction accuracy of POIs, it makes the network connection qualities on those POIs change dramatically. Consequently, the prediction of network connection quality based on the historical trace data will not be accurate. As shown at the 10000th episode, the numerical rewards of all algorithms decrease, especially to the proactive DDQN, which highly depends on the prediction results. When the uncertain events occur, the proposed proactive approach will still select the group of devices that are going to suffer from low connection-quality based on the false prediction. However, those candidates cannot complete the FL training tasks at all in the subsequent timeslots. This is why the reward of DDQN experiences a drastic drop of numerical reward after the uncertainties. Although DDQN still selects those group of devices at the very beginning exactly when the uncertainties occur, the Q-network quickly perceives the decreasing rewards. As a consequence, DDQN changes the policy selection adaptively towards higher rewards. Finally, DDQN achieves a high converged reward that is still close to the offline optimal performance. This result shows the advantage of DDQN algorithm when performing the distributed Federated Learning in a highly dynamic computing environment.

F. Ratio of Valid Participants

Fig. 10(a) shows the performance of all 4 algorithms in terms of the accumulated number of valid participants during the rounds of FL training. Recall that we call a candidate a *valid participant* if it can successfully report its updated

local model to the FL server within a round of FL training. In each algorithm, the dark bar shows the accumulated number of valid participants, while the shallow part indicates the number of invalid ones. We see that FedAvg has the largest number of invalid participants. The DDQN-based algorithm always shows the highest number of valid participants. Fig. 10(b) illustrates the ratio of valid participants under each algorithm. Once again, we observe that the proposed DDQN-based algorithm achieves a ratio as high as 95%. More clearly, FedAvg leads to a very low ratio of valid participants during all the FL training rounds. These results prove that the proposed proactive candidate-selection algorithms can always select the best group of candidates for training the global FL model.

G. Test Accuracy of Global FL Model using 2 Open Datasets

In the experiments using the non-i.i.d. dataset, each participant only holds 20%-40% of all image samples as their training datasets. The number of images in each participant's dataset is varying from 2500 to 5000. We then use a thread to simulate a mobile candidate, which adopts the preprocessed trajectory as its mobility data. Those candidates participate in the training of a global model residing in the FL server using their local non-i.i.d. image datasets. Note that, we don't evaluate the test accuracy of FedAvg algorithm, since FedAvg always selects all the K=10 candidates as participants. Thus, it is unfair to compare its test accuracy with other algorithms.

Fig. 11 demonstrates the time-varying test accuracies evaluated in each participant device after model aggregation on the FL server. We can observe that different candidate-selection algorithms yield different test accuracies. For example, in Fig. 11(a), the proposed DDQN-based algorithm shows a very close performance with the optimal *offline* scheme, and 10%-20% higher time-varying test accuracies than that of the reactive baseline FedCS. This is because the ratio of valid participants is higher under the proposed DDQN-based algorithm than that of FebCS. Thus, the FL server can receive much more updated model parameters and weights, which result in higher test accuracies of the aggregated global model. Similar results can be observed from Fig. 11(b). The reasons are same and thus omitted here for simplicity.
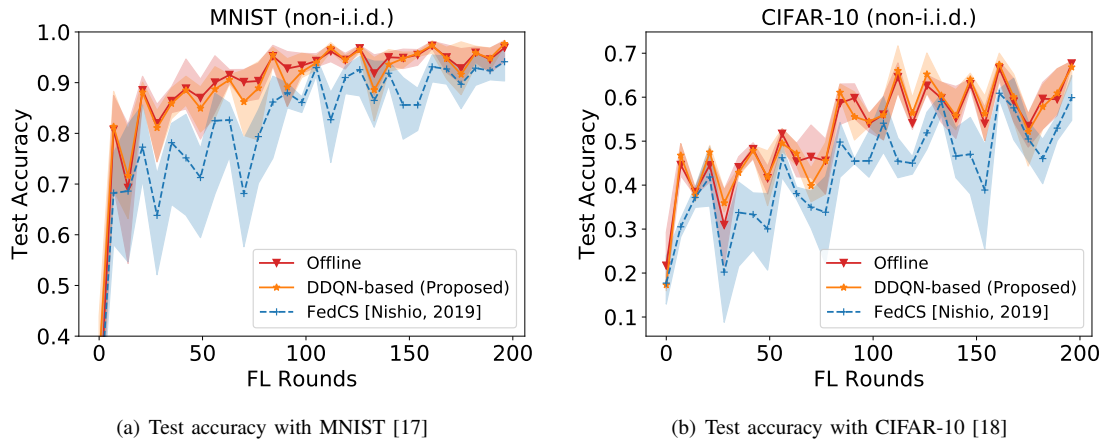
(a) Test accuracy with MNIST [17]

(b) Test accuracy with CIFAR-10 [18]

Fig. 11. Test accuracies of algorithms under the non-i.i.d. datasets of MNIST [17] and CIFAR-10 [18].

VII. CONCLUSION

Candidate-selection is the first essential phase during each round of the synchronous-fashioned Federated Learning. To the best of our knowledge, we are the first to propose the proactive candidate-selection by predicting the qualities of both training and reporting phases. The proposed DRL-based candidate-selection algorithm can adaptively handle the unexpected dynamic events occurred in the metropolitan mobile edge computing environment. To evaluate the effectiveness of the proposed proactive approach, we first collected the real-world user traces using a self-developed Andriod App. The trace-driven experiments show that the proposed DDQN-based algorithm can yield a near-optimal reward, and outperform other baselines with respect to multiple metrics including the ratio of valid participants and the ability to adapt to highly uncertain events. Using two open image datasets, experiment results show that the test accuracies of the proposed DDQN-based algorithm are higher than that of the existing candidate-selection baselines by around 10%-20% during the online FL training.

We believe this work will shed new lights on the synchronous-fashioned Federated Learning. The proposed DDQN-based algorithm will be viewed as a reliable approach that provides the proactive candidate-selection mechanism to the Federated Machine Learning.

REFERENCES

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," in *Proc. of the 20th International Conference on Artificial Intelligence and Statistics (AISTATS'17)*, 2017.

[2] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konecny, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[3] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[4] M. Chen, Z. Yang, W. Saad, C. Yin, H. V. Poor, and S. Cui, "A joint learning and communications framework for federated learning over wireless networks," *arXiv preprint arXiv:1909.07972*, 2019.

[5] M. C. Gonzalez, C. A. Hidalgo, and A.-L. Barabasi, "Understanding individual human mobility patterns," *nature*, vol. 453, no. 7196, p. 779, 2008.

[6] J. Feng, Y. Li, C. Zhang, F. Sun, F. Meng, A. Guo, and D. Jin, "Deepmove: Predicting human mobility with attentional recurrent networks," in *Proc. of the 2018 World Wide Web Conference (WWW'2018)*, 2018, pp. 1459–1468.

[7] Z. Zhou, X. Chen, W. Wu, D. Wu, and J. Zhang, "Predictive online server provisioning for cost-efficient iot data streaming across collaborative edges," in *Proc. of the 20th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019, pp. 321–330.

[8] R. Jiang, X. Song, D. Huang, X. Song, T. Xia, Z. Cai, Z. Wang, K.-S. Kim, and R. Shibasaki, "Deepurbanevent: A system for predicting citywide crowd dynamics at big events," in *Proc. of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*, 2019, pp. 2114–2122.

[9] D. Liao, W. Liu, Y. Zhong, J. Li, and G. Wang, "Predicting activity and location with multi-task context aware recurrent neural network." in *International Joint Conferences on Artificial Intelligence (IJCAI)*, 2018, pp. 3435–3441.

[10] Q. Lv, Y. Qiao, N. Ansari, J. Liu, and J. Yang, "Big data driven hidden markov model based individual mobility prediction at points of interest," *IEEE Transactions on Vehicular Technology (TVT)*, vol. 66, no. 6, pp. 5204–5216, June 2017.

[11] H. Shi, C. Zhang, Q. Yao, Y. Li, F. Sun, and D. Jin, "State-sharing sparse hidden markov models for personalized sequences," in *Proc. of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD'19)*, 2019, pp. 1549–1559.

[12] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017, pp. 1175–1191.

[13] T. Nishio and R. Yonetani, "Client selection for federated learning with heterogeneous resources in mobile edge," in *IEEE International Conference on Communications*, 2019, pp. 1–7.

[14] "Technique report," Jan. 2020. [Online]. Available: https://drive.google.com/file/d/1NhvbznAcmrWiacTAx6d7F4A7o0KWo5mu/view?usp=sharing

[15] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[16] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Thirtieth AAAI conference on artificial intelligence*, 2016.

[17] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. of the IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov 1998.

[18] A. Krizhevsky, "Learning multiple layers of features from tiny images," *University of Toronto*, May 2012.